## Remarks

This case has been carefully considered in light of the Office Action dated March 16, 2006 wherein: claims 1-3 and 5-13 were rejected under 35 USC 112, first paragraph; claims 1-3, 5-6 and 11-13 were rejected under 35 USC 102(b) on Kanfi; claims 7-10 were rejected under 35 USC 102(b) on Uemura et al.; and claim 6 was rejected under 35 USC 103(a) on Sarkar in view of Kanfi. Reconsideration is respectfully requested.

Claims 1, 6, 7, 11, 12 and 13 have been amended to recite the provision of the capability for application programs to determine "changes to said file by [reading] an incremental read of only blocks of said file that contain new data." This amendment is made to explicitly recite terminology that is used throughout the specification for clarity. No new matter has been added. Further to this point, the Applicants cite this specific language that appears throughout the specification below with respect to the traversal of the rejection under 35 USC 112.

The additional amendment to claim 1 that was made in the response dated March 6, 2006; i.e., incorporating the limitations of cancelled claim 14, was inadvertently made incorrectly and in error. In particular, the amendment was made without underlining. In addition, the cancellation of claim 4 was also made in error. This error occurred during the drafting of the amendment and was not corrected. The error has been corrected herein by removing the limitation that was added incorrectly and inadvertently and by adding a new claim 14 corresponding to original claim 4.

Claim 7 has been amended to recite "***an incremental read of only blocks of said file that contain new data , the incremental read of said file returning an indication of a hole for each portion of the file not containing data specified by said user, such that said write request inserts holes into said backup file, thereby bringing said backup file up to date.***" Support for this amendment is found, for example, in paragraph [0065]. No new matter has been added.

The Applicants respectfully traverse the rejection under 35 USC 112 and request withdrawal thereof for the following reasons.

In the Office Action, it is stated in the paragraph coupling pages 2 and 3 that "[i]t is not clear from the specification of this application as filed that how it is determined which block(s)

of the file is/are modified without reading the entire file? The entire file has to read/scanned in order to determine which block(s) is/are modified or contains new data in it." This is incorrect and goes to a critical aspect of the Applicants' invention. The Applicant respectfully submits that providing the capability "to determine changes to said file by an incremental read of only blocks of said file that contain new data", as recited in the claims, as amended, is described with respect to various embodiments of the Applicants' invention. To this point, the Applicants reference hereinbelow several examples from the specification.

The Applicants respectfully submit that the write request records the data as "changed" by updating a dirty bit or pointers to the data. This allows the read request to determine the changes ***by examining the dirty bits or data pointers, not the data itself.*** For example, as set forth in the specification on page 13, paragraph [0057], the Applicants describe this aspect of their invention as follows:

> In one embodiment of the present invention, the file system maintains the timestamp as a single "dirty" bit for each disk block assigned to the file. This bit provides an indication that the disk block does not currently contain valid data. The dirty bit may be stored within the inode file entry and/or within indirect blocks along with the disk pointers. Allocating or de-allocating a disk block as well as writing to an existing block sets the dirty bit. The extended read command of the present invention accesses the dirty bits to determine the changes and to reset the bits as the data is copied. For example, consider the situation in which the data is being read by a backup utility. ***The first time the backup utility runs, it copies all of the non-zero data and resets all of the dirty bits.*** The data is copied to another location, perhaps to a tape or to another file system located elsewhere. ***The next time the backup utility runs, it needs to read only the data that has changed since the first, original copy. The blocks that have changed are identified via the dirty bit.*** While reading the data, the dirty bits are reset and thus the file is ready to collect the changes for the next incremental backup. [emphasis added]

Another embodiment is described in paragraph [0060] on page 14 of the specification as follows:

> A preferred embodiment of the present invention utilizes a file system that supports snapshots, such as IBM's General Parallel File System (GPFS). The "copy-on-write" method used to maintain the snapshot also serves to identify the changed blocks in each file. ***The extended read command herein need only examine the intervening***

***snapshots to determine the incremental changes to the file.*** [emphasis added]

The embodiments of Figures 6A and 6B also illustrate this aspect of the Applicant's invention, as explained in the specification. In particular, in paragraph [0063] on page 15, Figures 6A and 6B are described as follows:

> ...Figures 6A and 6B illustrate the use of "dirty bit" indicators 321A, 321B, 321C and 321D as an example of one mechanism for controlling data status on a block-by-block basis, especially for file backup writing purposes. Figure 6A shows an initial state in which all of the dirty bits are reset to zero meaning that the data has not been modified. Figure 6B illustrates a file system data structure for the same file for the case in which new data has been written to data blocks 310B and 310D. In this case it is to be particularly noted that dirty bits 321B and 321D are now set at "1" to provide an indication that the data in the referenced blocks has been changed. Note that Pointer C still points to sparse data. In the example shown in Figures 6A and 6B, an extended read of the original "before" file returns the non-zero data in blocks 310A, 310B and 310D (since block 310C is null). ***An incremental read of the "after" file returns data for blocks 310B and 310D only.*** [emphasis added]

The embodiments of Figures 7A and 7B further illustrate this aspect of the Applicant's invention, as described in the specification. In particular, in paragraph [0065] on page 16, Figures 7A and 7B are described as follows:

> Figures 7A and 7B should also be considered together. These figures also show "before" and "after" views, respectively. Initially, all of the dirty bits are "clean." However, Figure 7B illustrates a scenario in which a new sparse region has been created and in which there is one new block of changed data. In particular, it is seen that Pointer B now reflects the fact that the previous data block (310B) is now sparse. Dirty bit 321B is set to "1" to reflect this change. At the same time, dirty bit 321D is set to "1" to reflect the fact that data block 310D has changed. In this example the "before" file has a hole in the third block (Pointer C) and data in blocks 310A, 310B and 310D. The drawings illustrates the situation that occurs if the file is truncated with respect to block 310B and new data is written to block 310D. Thus the "after" file now has a hole in blocks 310B and 310C, with the dirty bits set for pointers B and D only. ***An incremental read of the "after" file provides an indication that a new "hole" exists in block 310 B and new non-zero data in block 310D. A backup program which takes full advantage of this information applies this increment to a previously saved version of the "before" file by using the extended write call to write the new "hole" for block 310B into a***

> ***previously saved file. It then uses the extended write or a regular write to change block 310D, thus bring the saved backup file up to date.*** [emphasis added]

Also in paragraph [0067] on pages 17-18 of the specification, the Applicants describe another embodiment as illustrated in Figures 8A and 8B. In particular, toward the bottom of page 17 and continuing onto page 18, the Applicants state the following:

> .....Note that ***the ditto addresses provide the mechanism for recording the incremental changes to a file.*** The presence of a ditto address in a snapshot file indicates that the data stored in that block has not changed during the snapshot increment. Thus, an incremental read of the changes to the active file system since Snapshot #17 returns only the data in blocks 310B and 310D. An incremental read of the changes to the active file system since Snapshot #16 returns the data in blocks 310B and 310D and furthermore indicates a new hole in data block 310C. The incremental read can also be applied between snapshot versions of the file. ***An incremental read of the changes to the file between Snapshot #17 and Snapshot #16 would return the data for blocks 310B and 310D only. Although not shown in the example, the incremental read can be applied to any pair of snapshots, regardless of the number of intervening snapshots.*** [emphasis added]

Also, in paragraph [0069], the Applicants state the following:

> All of the methods described above to detect and record changes as well as sparse regions are fast and efficient. These methods are not heuristic solutions. Instead they exactly define the blocks that have changed. ***The extended read command determines the changed blocks by scanning the file's metadata and does not need to scan the actual file data.*** The preferred embodiment requires no additional storage for data signatures or time stamps, beyond the storage already required to implement the snapshot command. Since the file system already maintains this data, the extended calls merely provide a means for a general user to obtain the incremental changes to his own files. The method is also useable with the entire file system to support full backup or mirroring. [emphasis added]

The Applicants have thus provided numerous references to the original specification and drawings which provide support for the amendments to the claims (i.e., "to determine changes to said file by an incremental read of only blocks of said file that contain new data"). Reconsideration and withdrawal of the rejection under 35 USC 112 are thus respectfully requested.

The Applicants traverse the rejection of claims 1-3, 5-6 and 11-13 under 35 USC 102 on Kanfi for the following reasons.

The Office Action quotes the first sentence of the abstract of Kanfi, to wit: "A facility is provided ***for storing*** in a backup memory only those blocks of a file, or disk partition, which differ from the corresponding blocks forming an earlier version of the file." [emphasis added]

Despite the arguable similarity between Kanfi and the Applicants' invention with respect to storing in backup memory only those blocks of a file that have changed, the Applicants respectfully submit that their invention as recited in their claims is patentably distinguishable over Kanfi in several major respects and thus avoids disadvantages of Kanfi. In particular, although Kanfi only stores the blocks of a file that have changed, Kanfi must generate a table of data signatures associated with the entire file, which must be stored and read before storing the changed blocks. This is in contrast to the Applicants' ***incremental reading***, as recited in the claims and described hereinbelow. Kanfi explains in column 4, lines 15-26:

> Assume that, after a period of time following the initial archiving of file F1, computer 10-1 communicates with computer 110 for the purpose of storing in one of the archive memories 30-1 through 30-P, e.g., memory 30-1, the latest version of file F1. In doing so, computer 10-1 generates a signature for each block forming the latest version of file F1 and stores each such signature in sequence in a table formed in the internal memory of computer 10-1. An example of the latter table is shown in FIG. 4.
>
> Following the foregoing, computer 10-1 then compares each entry in the newly formed table 400 with its corresponding entry in previously formed table 200.

In contrast, the Applicants have provided a method for tracking incremental changes to a large and/or sparse file by providing the capability for application programs ***"to determine changes to said file by an incremental reading of only blocks of said file that contain new data,"*** as recited in the amended claims. The Applicants advantageously avoid the drawbacks of using heuristic data signatures, as Kanfi does, which must be stored or regenerated with each backup, and which still require the reading of an entire file, as explicitly pointed out by the Applicants in the specification, e.g., in paragraphs [0004], [0006]-[0008] and [[0069]. As recited in the claims, as amended, the Applicants advantageously avoid these drawbacks (e.g., reciting in

the claims as amended ***"to determine changes to said file by an incremental reading of only blocks of said file that contain new data,"***), such that the Applicants' claims 1-3, 5-6 and 11-13 are believed to be patentably distinguishable over Kanfi under 35 USC 102.

The Applicants also respectfully traverse the rejection of claims 7-10 under 35 USC 102(b) on Uemura et al. for the following reasons.

The Applicants disagree with the Examiner that Uemura teaches a method for backing up sparse files, which is the subject of claims 7-10, as amended. The Applicants respectfully submit that the differences between "incremental backup" and "backing up sparse files" is well-known in the art and is highlighted in the specification, for example, in paragraph [0027] where it is stated that "....another embodiment provides a method for retrieving all of the non-zero data in a sparse file (as opposed to the incremental changes only)."

However, even assuming *arguendo* that Uemura teaches a method for backing up sparse files, as suggested in the Office Action, the Applicants respectfully submit that claims 7-10, particularly as amended, are patentably distinguishable from Uemura under 35 USC 102 for the following reasons.

Uemura describes an incremental backup system, including a storage unit that is accessed in block units of predetermined size for storing data to be backed up. Uemura uses difference map information to record the latest backup generation number to indicate when data in each block has been updated. Uemura further uses a management mechanism for managing backup generation numbers for each block.

In contrast to Uemura, the Applicants recite in amended claims 7-10 "[a] method for backing up sparse files and tracking incremental changes thereto, said method comprising the step of:

> writing to a backup file in a write request to a file system in which at least one user specified portion of said file is defined to have a specified value and in which the size of said at least one portion is specified by said user , said write request providing the capability for application programs to ***determine changes to said file by [reading] <u>an incremental read of only blocks of said file that contain new data, the incremental read of said file returning an indication of a hole for each portion of the file not</u>***

> ***containing data specified by said user, such that said write request inserts holes into said backup file, thereby bringing said backup file up to date.*** [emphasis added]

Uemura do not deal in any way with specific handling of holes in sparse files. That is, in contrast to claims 7-10, particularly as amended, Uemura do not teach or suggest an incremental read of a file "***returning an indication of a hole for each portion of the file not containing data specified by said user, such that said write request inserts holes into said backup file, thereby bringing said backup file up to date***" Therefore, claims 7-10, particularly as amended, are believed to be patentably distinguishable from Uemura under 35 USC 102.

The Applicants also respectfully traverse the rejection of claim 6 under 35 USC 103 on Kanfi in view of Sarkar for the following reasons.

The Office Action states the following on page 9:

> Examiner agreed with Applicant that in Sarkar reference entire file needs to be scanned in order to come up with which are changed since last backup, however, ***Kanfi teaches that the write request providing the capability for application programs to determine changes to the file by reading only blocks of a file that contain new data*** (e.g., "A facility is provided for ***storing*** in a backup memory only those blocks of a file, or disk partition, which differ from corresponding blocks forming an earlier version of the file" see the abstract). Accordingly, it would have been obvious to one of ordinary skilled [sic] in the art at the time of the current invention was made to implement Kanfi's teachings in the method taught by Sarkar. ***In doing so, (i) the backup process will be faster and more efficient since only the new data would be backed up instead of the whole file; and (ii) the storage space of the backup storage will be reduced since only the new data will be backed up.*** [emphasis added]

The Applicants respectfully point out that the Office Action has not only mischaracterized Kanfi in making this rejection, but has furthermore missed the critical aspect of the invention set forth in the limitation ***"to determine changes to said file by an incremental reading of only blocks of said file that contain new data,"*** as recited in the claims as amended. In particular, although Kanfi stores only the changed files in backup memory, as indicated in Kanfi's Abstract as quoted by the Examiner, Kanfi requires storing a file of data signatures and reading the entire file to determine what the changes are. In contrast, the Applicants "***determine changes to said file by an incremental reading of only blocks of said file that contain new data.***" This critical

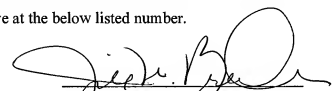
advantage over Kanfi (***an incremental read of*** only the changed data) has been equated in the Office Action to ***storing*** only the changed data, without reference to how data is read and compared by Kanfi. This is not correct for all the reasons set forth hereinabove with respect to traversing the 102 rejection over Kanfi.

In addition, as pointed out on page 9 of the Office Action, the Examiner agrees with the Applicants that in Sarkar, the "entire file needs to be scanned in order to come up with which are changed since last backup."

Hence, neither Sarkar nor Kanfi, alone or in combination, teach or suggest an incremental read of only blocks of the file that contain new data, as recited in the Applicants' amended claims. Indeed, Kanfi requires storing, reading and comparing a table of data signatures; and Sarkar requires reading the entire file. Therefore, any combination of Sarkar with Kanfi would not result in the Applicants' invention, as recited in the claims, particularly as amended. That is, neither Kanfi nor Sarkar, alone or in combination, would render obvious the Applicants' invention, including the limitation about determining ***changes to said file by an incremental reading of only blocks of said file that contain new data.***

Therefore, claim 6 is believed to be patentably distinguishable from the suggested combination of Sarkar and Kanfi under 35 USC 103.

Should the Examiner have any further concerns regarding this application, he is invited to contact Applicants' representative at the below listed number.



Jill M. Breedlove
Attorney for Applicants
Registration No.: 32,684

Dated: July 17, 2006

HESLIN ROTHENBERG FARLEY & MESITI P.C.
5 Columbia Circle
Albany, New York 12203-5160
Telephone: (518) 452-5600
Facsimile: (518) 452-5579